Container Hardening Guide

Version 1, Release 1

15 October 2020

Developed by DISA for the DoD

Trademark Information

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

1. OVERVIEW

1.1 Introduction

This document focuses on the Department of Defense (DoD) Enterprise DevSecOps Initiative (DSOP) and was created to detail the Enterprise DevSecOps Container Hardening Process and ensure it meets the DoD Hardened Containers Cybersecurity Requirements. It is important to understand both DevSecOps and cybersecurity concepts and principals, as well as have knowledge of containers and container platforms. Refer to the Master Approach Document for more information on how the DSOP platform functions.

DevSecOps is a set of software development practices that combines software development (Dev), security (Sec), and information technology operations (Ops) to secure the outcome and shorten the development lifecycle. Software features, patches, and fixes occur more frequently and in an automated fashion. Security is applied at all phases of the software lifecycle. An example DevSecOps process is shown in Figure 1-1 below. This document will focus on the process of securing containers that are placed into a common repository for everyone to use. This repository can be found at https://repo1.dsop.io/dsop (see Figure 1-2).

Note: This document focuses on container security. It is understood that any application code or library must be scanned by static/dynamic code analysis tools and passed or have mitigated/accepted the risk prior to being integrated into a container for DoD use. If the application is already approved for use (and scanned) by IC/NSA/DoD CIO/DISA, reciprocity can take effect. This document does not describe that process.

Figure 1-1: DevSecOps Example

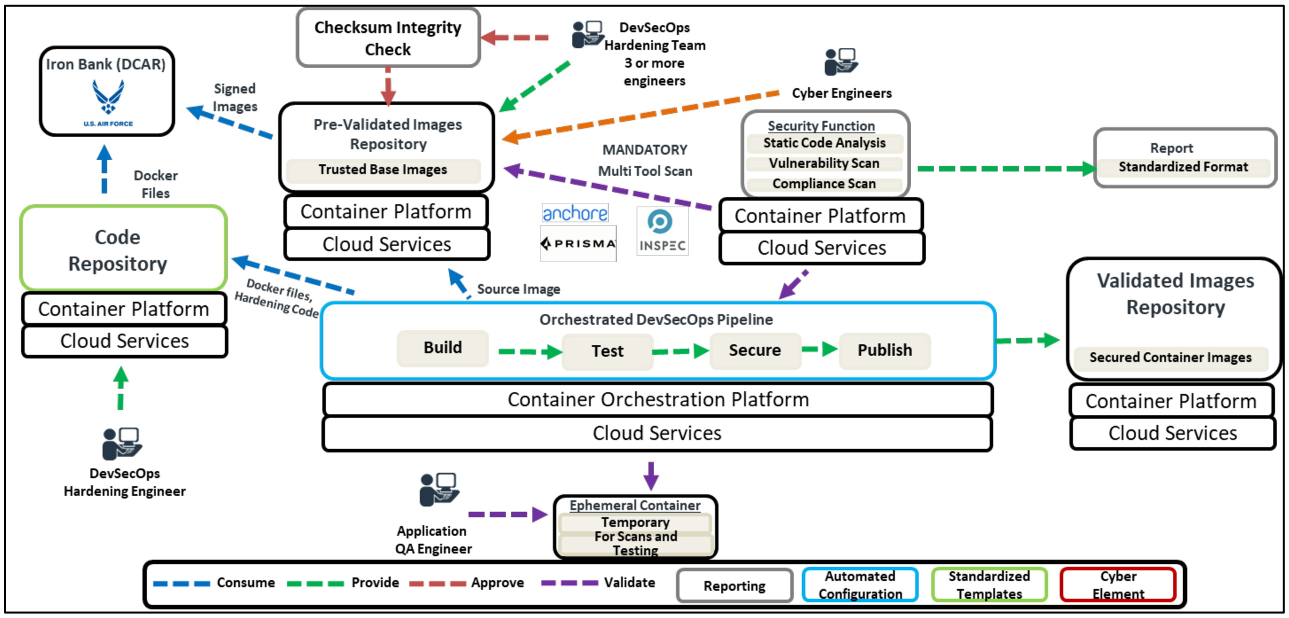
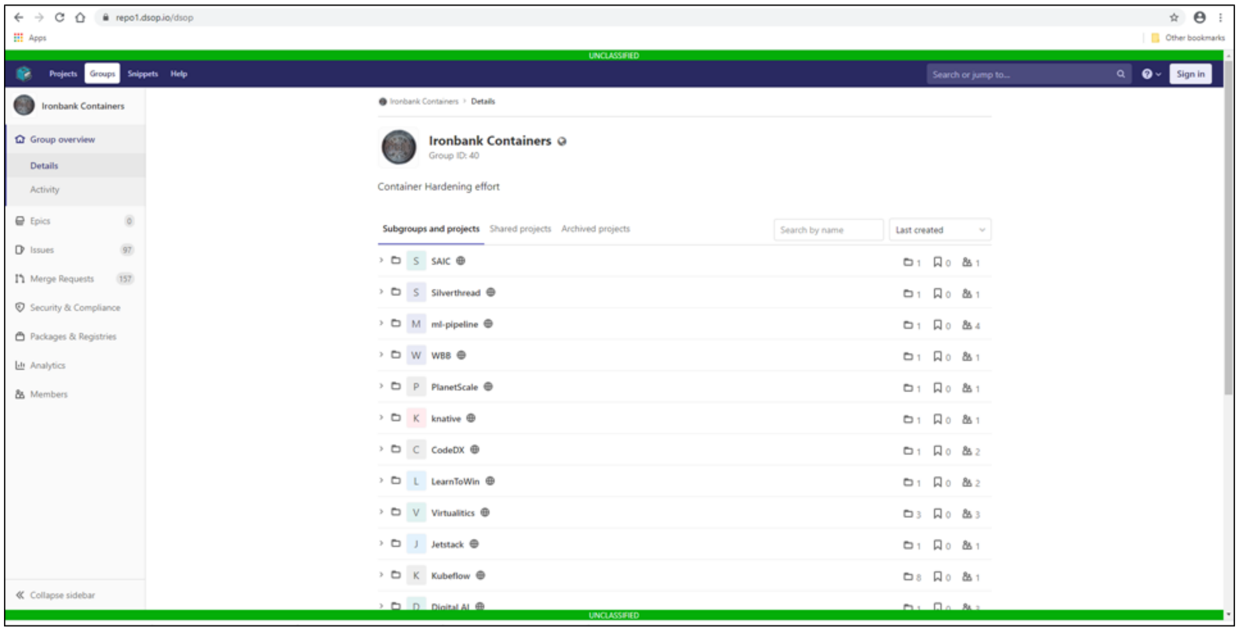


Figure 1-2: Iron Bank Repository



Ironbank Containers
Group ID: 40

Container Hardening effort

S	SAIC	1	0	1
S	Silverthread	1	0	1
M	ml-pipeline	1	0	4
W	WB8	1	0	1
P	PlanetScale	1	0	1
K	knative	1	0	1
C	CodeDX	1	0	2
L	LearnToWin	1	0	2
V	Virtualitics	3	0	3
J	Jetstack	1	0	1
K	Kubeflow	8	0	1
D	Digital AI			

1.2 Definitions

In describing the hardening process, it is necessary to define concepts and terminology. This section will define and describe items discussed in the process as well as other sections of this document.

1.2.1 DoD Hardened Containers (DHC)

A DoD hardened container is an Open Container Image (OCI)-compliant image that is secured and made compliant with the DoD Hardened Containers Cybersecurity Requirements. Container images should adhere to the OCI Image Format Specification to ensure portability whenever possible. These containers are made available for use in the Iron Bank, the centralized repository.

1.2.2 Container Hardening Team (DHT)

The Container Hardening Team is responsible for hardening DoD containers according to DoD Hardened Containers Cybersecurity Requirements. The team is composed of DevSecOps Engineers and other container experts that have knowledge of the product being hardened. They also have an understanding of the DHC Cybersecurity Requirements and DISA Security Requirements Guide (SRG) and Security Technical Implementation Guide (STIG) information. A single team member is called a Hardener.

Hardeners also require the following skills:

- Expertise in OCI-compliant container images, including creating compliant Dockerfiles

- Understanding of DISA STIG/SRG documentation as it applies to the application run
- Expertise in the DHC Cybersecurity Requirements
- Strong understanding of Container security solutions (Prisma, Anchore, and StackRox) for vulnerability scanning, Chef InSpec and OpenSCAP for compliance scanning.

1.2.3 DoD Container Factory (DCF)

To automate the hardening process, the DoD has implemented a DSOP platform on GovCloud at Impact Level 2 (publicly accessible and open-source) to set up the Continuous Integration/Continuous Delivery (CI/CD) pipeline leveraged to automate the container hardening process. This DevSecOps process is similar to Figure 1-1 and will be used primarily for the validation of hardened containers.

1.2.4 Repo One/DoD Centralized Container Source Code Repository (DCCSCR)

This is the web location used to store the hardened Dockerfiles and associated documentation. This DoD enterprise source code repository will be centrally hosted so Hardeners can store their code and leverage a CI/CD pipeline.

For onboarding containers into Repo One, view the prerequisite information shown at https://repo1.dsop.io/dsop/dccscr/tree/master/contributor-onboarding or view the process detail in Section 2.

1.2.5 Iron Bank/DoD Centralized Artifacts Repository (DCAR)

This is the system that stores the DoD Hardened Containers and associated documentation. This DoD enterprise centralized artifacts repository is implemented on a DoD Approved Cloud to provide access to DoD Programs and to the DoD Hardened Containers.

The artifact repository supports both files (traditional artifacts) and containers, as well as container registry capabilities. It provides a secure mechanism to store, track, sign, and distribute approved containers. It is accessible at: https://ironbank.dsop.io.

1.2.6 DoD Approved CI/CD Orchestration Solution (DACCOS)

In a DevSecOps system, a CI/CD pipeline is integral, and an orchestration solution is needed. The DoD has chosen CI/CD orchestration solutions such as Jenkins (open source), Cloudbees Jenkins, and GitLab for this function, and all are containerized and hardened where applicable.

1.2.7 DoD Base Container Image Approved Source (DBCIAS)

To ensure container base images are from approved sources, a container base image must be downloaded from a DBCIAS as a baseline.

There are two types of repositories: trusted and untrusted. Trusted repositories are used as an approved source for downloading container base images. Untrusted resources cannot be used in this fashion unless a Memorandum of Understanding (MOU) is reached between the DoD and the vendor or mission partner using the untrusted source. The MOU will dictate strict levels of security and ensure access to base images and their updates.

The following sources are currently approved and should be used in order of priority:

- Iron Bank/DCAR (approved DoD-wide; trusted)
- Product vendor proprietary repository (untrusted)
- Docker Hub (untrusted)
- Red Hat Container Repository (untrusted)

A trusted repository indicates that the containers were hardened by the DHT and can be used as is. See Section 2.1 for the vendor/mission partner prerequisite review steps before selecting a base image.

2. CONTAINER HARDENING PROCESS

The container hardening process requires the mission partner work closely with the DoD Container Approvers and Hardeners so that cybersecurity confidence can be gained about the containers prior to DoD use.

2.1 Container Hardening Prerequisite Steps

Some key prerequisite steps are listed below. The DHT is available to assist all personnel if there are questions.

Tech Note: A Dockerfile execution must begin with a 'FROM' instruction. This may be after parser directives, comments, and globally scoped ARGs. As an argument, a name is provided with an optional tag for an image that should be used as the base for the Docker container. Instead of building a host OS from scratch, the FROM instruction allows a base image for Linux or another OS as the Dockerfile basis.

1. Always prioritize the reuse of a Dockerfile from Iron Bank. Alternatively, a Dockerfile from its vendor or open-source community can be reused if it can be switched to an approved DoD base image. Ideally, the team should ensure the Iron Bank's accepted base images (UBI7/UBI8/scratch/distroless) are used by upstream partners/vendors.
2. Ensure that the Dockerfile uses a DoD-hardened/approved base container whenever available. Use the FROM command to do this. For example, if the Jenkins Dockerfile uses the Docker Hub OpenJDK base image, ensure the FROM instruction is replaced by the DoD-hardened OpenJDK base image instead. Use of the STIGed DoD-approved Universal Base Image (UBI) is recommended, as is the use of the approved DoD scratch and distroless image. These are open source, free, and available on the Iron Bank.
3. Use base images that reuse the closest hardened image layer available to inherit its hardening. For example, if a Java application is used, use the base image of the DoD-hardened OpenJDK and not just the DoD-hardened UBI image. It is highly recommended to have the DoD Enterprise DevSecOps team review and agree to the selection of a base image if it does not come from the Iron Bank.
4. If there are multiple source options/tags/variants for the base image chosen, ensure the image is scanned with both Prisma (or StackRox) and Anchore, to understand security risks associated with each image. For example, if you can download the image from Docker Hub, RedHat registry or Oracle registry, select the image most closely aligned with the DoD Hardened Containers Cybersecurity Requirements based on the scan results of those images.
5. If the base image has security flaws such as critical vulnerabilities, attempt to mitigate the flaw by applying security hardening, configuration changes etc. If that does not correct the flaw, the library/binary impacted must be removed or the risk documented as accepted.
6. Prepare all dependencies in the Repo One prior to submitting for hardening. All dependencies that need to be downloaded from the internet must be included in the Download.json or Download.YAML file to allow the container hardening system to build containers offline. If a container can benefit the DoD, then it can be considered for

inclusion in Repo One or another repository compliant with the DoD Enterprise DevSecOps Reference Design. The platform and hardened containers must be compliant with the minimum viable product requirements or it cannot be accredited.

7. It is critical for containers to be able to be built on classified and air gapped environments. Dockerfiles should not download any artifacts. Instead, a pre-phase step called Download will occur so the platform can download the required dependencies online. Then, the rest DoD Container Hardening process will operate in a disconnected and/or classified environment that does not have internet access. This will ensure the process will not be able to pull files from the internet that are required to build.

Tech Note: Platform One spends a great deal of time to ensure dependencies are not downloaded without their knowledge. Initially, Platform One will list all dependencies, download them while connected to internet, go Offline, and build the container validating the dependency downloads are not attempted. This critical security process ensures decoupling the download from the build process.

8. If an organization that has a capability, has two license options (for example, open source and an enterprise managed version), both containers must be provided to Repo One. The organization cannot only provide the paid version. The DoD Hardening Team will not accept enterprise containers if the open-source container is not also provided.

2.2 Container Hardening Process

The process described in this section is for vendor or mission partner informational awareness. It is important for mission partners and vendors to understand the best practice considerations shown in Appendix D: Container Security/Remediation Considerations. It is also important to understand that as containers inherit many of the security controls from the underlying Platform as a Service (PaaS) solution, the underlying host OS must be hardened per Agency-specific security guidelines as listed in Appendix C: Security Control Inheritance.

It is important to understand that containers must run on a DevSecOps Platform compliant with the DoD Enterprise DevSecOps Ref Design and its MVP requirements, including the Sidecar Container Security Stack, which enforces behavior detection and zero trust in runtime.

Without a CNCF-compliant Kubernetes platform and the MVP Ref Design capabilities, a standalone container is not accredited and its certificate to field does not apply.

The container hardening and scanning pipeline is located at https://repo1.dsop.io/dsop. Note that third parties are not allowed master branch write access to the Repo One repository; only members of the Hardening Team are authorized to make changes that will affect hardened containers including, Dockerfiles, readme files, license files, and anything else required for the scanning and hardening of the container. A Container Hardener will approve the container into the Repo One and Iron Bank repository after validation and approval. This process should also support free and opensource software (FOSS) as well as commercial off-the-shelf (COTS) software.

Using the container software factory or pipeline, the Hardener will begin the hardening process as depicted in the Figure 7-1 sub-process titled "Begin Hardener process". The detailed steps and examples are shown in Table 2-1 below.

Table 2-1: Hardener Process

Step	Process Description	Example	Resource
1	Create a new folder in Repo One under the organization name.	See Appendix A: Folder Structure Example. If open source, list as open source.	Hardener
2	Set up a new folder in the Repo One under the product name. **Note:** If a product has multiple variant options such as "cli" mode or "daemon" mode or x86 versus x64 versions, various subfolders can be created for each variant.	See Appendix A: Folder Structure Example.	Hardener
3	Define and list which version of the product and hardened containers will initially be supported.	If a Hardener needs to harden "Jenkins" with UBI base image, and the current OSs supported on Docker Hub are 2.59 to 2.60, the Hardener should start with 2.59 and move to the latest version, ideally using the same documentation with attempted reuse of the same Dockerfile created across versions. **Note:** Hardening a different OS requires the OS to be STIGed! Use STIGed UBI OSs if possible.	Hardener
4	Select a DoD Base Container Image Approved Source to download the base image from for each version.	Iron Bank is the DoD trusted repository, for example.	Hardener
5	Set up a subfolder within Repo One for each new version that will contain the newly created Dockerfile and related scripts/documentation/LICENSE/README.md.	Under "Open-source" and under "Jenkins" a subfolder ("2.59", "2.60", etc.) will be created per version. See Appendix A: Folder Structure Example.	Hardener
6	Within each version, starting with the lowest version supported, create a Dockerfile in Repo One that will use the selected base image. The	To understand which instructions are required, the Hardener will make decisions based on the Scanner	Hardener

Step	Process Description	Example	Resource
	Dockerfile will download the appropriate base image. The Dockerfile will also have additional instructions used to harden the container based on the Hardener decisions to mitigate findings and/or ensure proper DoD compliance.	findings from the scan process. See Section 2.3.	

Note: It is imperative to reuse instructions as much as possible between versions so hardening can be leveraged across versions. It is understood that new versions bring additional or new features, which can require additional hardening. Therefore, the Dockerfile should be reused from version minus one (if it exists) to avoid redundant work and leverage prior work.

2.3 Container Scanning Process

Using the CI/CD orchestration tool, the Repo One folder content will be downloaded into the pipeline and the Dockerfile used to build the container using Docker. This is done by reusing the Jenkins file.

Table 2-2: Scanning Process Steps

Step	Process Description	Example	Resource
1	The CI/CD pipeline will be triggered once the files are placed in the Repo One folder. Health and readiness will then be assessed.		Hardener
2	The pipeline will automatically begin to run the necessary scanners connected to it for compliance and vulnerability checks. It is mandatory to execute both Anchore and Prisma/StackRox checks. For Atomic OpenSCAP or Docker OSCAP, one must be run.		System
2a	Repo One Prisma/StackRox– BSH compliance rules will be loaded into the product for policy checks. Using the containers hardened from Repo One is mandated to ensure the proper policies are loaded and updated. This step is mandated.	Prisma/StackRox configuration must set SCAP_ENABLED to true. Scanner must have updated vulnerability information prior to every run.	Hardener

Step	Process Description	Example	Resource
2b	Repo One Anchore – JSON xml compliance rules must be loaded into the product for policy checks. Using the containers hardened from Repo One is mandated to ensure the proper policies are loaded and updated. This step is mandated along with Prisma/StackRox.	Custom policy checks can be made. Scanner must have updated vulnerability information prior to every run.	Hardener
2c	Atomic OpenSCAP – This product will operate on the command line to provide additional CVE and compliance scans. If Atomic OpenSCAP is not used, Docker OSCAP must be used.	The focus is to manage Atomic host system and containers. For additional information, see "Atomic vs. Docker-OSCAP Whitepaper" by DISA.	Hardener
2d	Docker-OSCAP – OpenSCAP provides vulnerability and compliance scanning for Docker images and containers. If Docker OSCAP is not used, Atomic OpenSCAP must be used.	The focus is on Docker images and containers. For additional information, see "Atomic vs. Docker-OSCAP Whitepaper" by DISA.	Hardener
3	If findings exist, continue to the mitigation process in Section 2.4; otherwise, continue to the approval process in Section 2.5.		Hardener

2.4 Findings Mitigation Reporting

Once findings are produced, it is necessary to report on these findings to the DevSecOps Approver as well as the mission partner or vendor. Table 2-3 lists some of the common recommendations made to mission partners and vendors.

Table 2-3: Mitigation Recommendations

Step	Process Description	Vendor/Partner Recommended Action	Resource
1	Iterate through the results, creating mitigation Dockerfile instructions for the mission partner or vendor.		
1a	Check base images, ports and list findings.	Enable or disable all ports except for those that are necessary to operate. Change operating ports to numbers other than standard 443 or 80.	Hardener

Step	Process Description	Vendor/Partner Recommended Action	Resource
1b	List lack of transmission security if it exists.	Ensure that data in transmission is secured using TLS. Strive for TLS v1.3.	Hardener
1c	List default configurations in use.	Disable unused features and modify default configurations to reduce the possible security breach footprint.	Hardener
1d	Emit logs to stdout.	Logs must be emitted to stdout or accessible locations.	Hardener
1e	User privileges	Do not run with elevated privileges. Root ID (UID 0) must not be used unless risk accepted and documented.	Hardener
1f	Decouple configurations to make them more portable. Use secrets for sensitive information such as tokens, keys, and passwords.	Use arguments, environment variables and configmaps for configurations and key value pairs. Also, use encoding such as base64 to obfuscate and encrypt yaml secrets.	Hardener
1g	User access	Integrate with the DoD hardened Keycloak SSO stack for single sign on capability or Require Common Access Card (CAC) access through SAML when possible. If not possible, MFA may be used based on risk acceptance.	Hardener
1h	Ensure secure connections are leveraged.	Use TLS and secure certificates when connecting with other services and containers. See 1b.	Hardener
1i	Remove unnecessary packages, services, and applications.	Remove anything not core to the operation of the container.	Hardener
1j	Remove User and Group permissions.	Remove the setuid and setgid permissions so that file permissions cannot be modified for malicious purposes.	Hardener
1k	Comply with DoD Hardened Containers Cybersecurity Requirements.	See Appendix B: DoD hardened Containers Cybersecurity Requirements.	Hardener
1l	Avoid command injections.	Be cautious with command injections in RUN, particularly when using ENV or ARGs.	Hardener

Step	Process Description	Vendor/Partner Recommended Action	Resource
1m	Leverage Kubernetes secrets	Ensure that Kubernetes Secrets and/or secrets vault are used to store secrets.	Hardener
2	Perform a container HEALTHCHECK to ensure proper overall functionality as well as the health and readiness of the container.		Hardener

2.5 Approving Process

From the mitigation reports, the DevSecOps Approver can determine if the results warrant approving the container. The four options are as follows: Rejected, Conditional Approval with Time Limit, Conditional Approval, and Approved. Each option is explained in more detail below.

- **Rejected** – The container has been reviewed and it was determined that it does not meet DoD standards for DoD enterprise-wide distribution.
- **Conditional Approval Time Limit** – The container will be approved provided the findings are remediated or mitigated within a configurable number of days, dependent upon the organization directive.
- **Conditional Approval** – The container has been approved with current mitigations in place.
- **Approved** – The container has been approved for distribution as is.

The Approver will reach a decision on each container via a process that depicts the number and severity of findings from the reporting performed in Section 2.4. For example, if the number of findings is high but most are high to low, the approver may reject the container altogether due to the magnitude of findings. If the Approver determines a critical finding exists that is key to keeping the DoD secure, the container may be rejected. Conversely, if the finding report dictates that the container only has a few low to medium findings, the container may be approved in its entirety.

The level of acceptable risk for cybersecurity findings that cannot be remediated at the time of processing will be decided on a case-by-case basis. These will all be reviewed by the Iron Bank or the PMO Authorizing Official (AO).

Regardless of the outcomes from the Approval process, all findings and recommendations are shared with the mission partner or vendor. If a container is placed in the Approval Time Limit process, the container is placed into a file with a trigger for re-run at X time and date. For any containers approved, the following steps will be performed.

Table 2-4: Approved Container Deployment Process

Step	Process Description	Example	Resource
1	The container is signed and checksum is created.	Container is signed with a DoD approved key and a checksum created using SHA 256 or more.	Hardener
2	Copy Helm/Operators and the README/LICENSE file		Hardener
3	Checksum and artifact storage	Store checksum and public key alongside artifact in Iron Bank.	Hardener
4	Store Repo One files in DCAR as pre-production.	Store files and tag version as pre-production until CtF, at which time it will be tagged as production.	Hardener
5	Scripts used for automation	Scripts used to automate deployment or facilitate adoption must be stored under a “scripts” folder.	Hardener
6	Configuration files for containers	Configuration files for specific container configurations must be stored under a “config” folder.	Hardener
7	Helm charts and Kubernetes Operators	Helm charts or Kubernetes operators should be stored in Iron Bank and Repo One to facilitate adoption.	Hardener
8	Additional documentation along with base container documentation	Documentation necessary to accompany container must be copied into both repositories under a “documentation” folder.	Hardener
9	README files	Vendor or Hardener should provide a README.md file within the subfolder that includes the impact level for which the container is intended.	Hardener
10	Scan results	Results from Scan tools will be copied into compliance folders as HTML files and placed into Iron Bank repository. As Prisma/StackRox and Anchore are widely available and can be run by anyone, the scan results are NOT considered FOUO.	Hardener
11	Risk Management	Risk Management documentation created by the Hardener to document findings, false positives, accepted risks, and current container risk posture will be stored in a subfolder	Hardener

Step	Process Description	Example	Resource
		called "accreditation" in both repositories.	
12	Changelog	Changelog will be added to both repositories' subfolders and updates as appropriate.	Hardener
13	Examples	Examples can be added to support container adoption. These will be stored in an examples folder in both repositories.	Hardener
14	Licensing	Each hardened container requires a file that describes the licensing required to run the container (LICENSE). This information (including files) should be duplicated in both repositories. README.md will also indicate if a product requires a commercial license.	Hardener

Note: Ensure any changes to the Repo One folder are changed in Iron Bank as well. If a Dockerfile changes, the new artifact must go through the hardening process again.

2.6 Automated Hardening

The hardening process should be fully automated. When a new release is detected by the pipeline, files are automatically downloaded from the repository and the steps are automatically run as listed in the prior process tables. The pipeline will be configured to search for updated dependencies/binaries and automatically run these with the pipeline, as well as notify the team of the build results. It is important for vendors and mission partners to note that files (artifacts/binaries) will be pulled from their existing agreed sources; it is not acceptable to host binary files on Repo One.

If a new CVE is detected in an upgrade, the build will stop and a manual assessment of the new CVE will be conducted.

2.7 Third-Party Container Sharing

Third parties must provide complete Dockerfiles, download.json/yaml file, scripts, and config files so new folder creation and ingestion into Repo One can take place. Binary container images will not be accepted and Dockerfiles must be provided to replicate the build.

It is allowed to leverage binaries (as long as they are downloaded by the Download.json or Download.yaml script) in Dockerfile for the application layer but the source of the Dockerfile must be provided so the container can be rebuilt. The goal is to integrate existing Dockerfiles into the DoD hardening process and Repo One so that new containers benefit from the same automation.

2.8 Keycloak Integration

Keycloak provides a centralized, delegated administrative model and single sign-on (SSO) capabilities to an organization. Keycloak leverages DoD PKI and stores information about objects on the network and makes this information available for administrators and users to find and utilize. Keycloak uses a structured data store as the basis for a logical, hierarchical organization of directory information. For DoD access, organizations need the ability to use the CAC as the principal card to enable physical access to buildings and controlled spaces as well as access to the DoD computer network and systems.

As Keycloak is fully integrated with the DoD ADFS/PKI, it is strongly recommended that any system operating in Impact Level 4 or higher use Keycloak for any user authentication. For non-person entity-to-entity communication, signed x509 certificates and PKI should be used.

3. APPENDIX A: FOLDER STRUCTURE EXAMPLE

3.1 Inside Repo One DCCSCR

Figure 3-1: Example 1: Repo One DCCSCR

Oracle					Vendor name
	MySQL				Product name
		v.8.0			Product version
			Dockerfile		File using the base approved image and hardening the container by using Dockerfile INSTRUCTIONS
			README.md		README file to describe the container, warn if this a commercial product or not, etc.
			Scripts		Scripts folder (optional)
				Script files	.sh or other scripts
			Config		
				Config files	Configuration folder
			documentation		Documentation folder (optional)
				PDF files	
			Compliance		
				SCAP or OVAL files	
			Accreditation		Accreditation folder (optional)
				PDF files	risk mgt. of the container
			Examples		Examples folder (optional)
				Misc. files	Documentation that shows how to use the container
			LICENSE		License text file describing license of the product; if commercial, a POC must be listed for vendor questions
			Changelog		Changes from previous version
			Download.yaml or Download.json	Rpm or download files	File listing everything to be downloaded from the internet to build the container; it can include application binaries
		v.5.7			
			Repeat		Replicate structure as v.8.0
		v.5.6			
			Repeat		Replicate structure as v.8.0

3.2 Inside Iron Bank DCAR

Note: Bold font shows custom files created by CI/CD pipeline that are not present in REPO ONE.

Figure 3-2: Example 2: Iron Bank DCAR

Oracle					Vendor name
	MySQL				Product name
		v.8.0			Product version
			mysql-v.8.0-hardened-container.tar		tar version of the hardened container built
			mysql-v.8.0-checksum		checksum of the untar version of the container
			README.md		README file to describe the container, warn if this a commercial product or not, etc.
			Config		Config folder optional
				script files	
			documentation		Documentation folder optional
				PDF files	
			compliance		Policy compliance folder
				result files	All HTML/PDF results from container security scanners
			accreditation		Accreditation folder optional
				PDF files	Risk management files
			examples		Example folder optional
				misc. files	Documentation that shows how to use the container
			license		License text file describing license of the product; if commercial, a POC must be listed for vendor questions
			changelog		Changes from previous version
			Helm charts or Kubernetes operators		Containers can include Helm charts and Kubernetes operators to facilitate adoption.
			v.5.7		
				repeat	Replicate structure as v.8.0
			v.5.6		
				repeat	Replicate structure as v.8.0

4. APPENDIX B: DOD HARDENED CONTAINERS CYBERSECURITY REQUIREMENTS

- Comply with initial and ongoing DoD Cybersecurity accreditation regulations/ frameworks.
  - The container base OS image must be STIGed assuming a STIG is available. If an OS STIG is not available (Alpine, for example), use the generic UBI STIG or DoD scratch or DoD distroless. It is understood that many of these STIGs were designed prior to the existence of containers. If a specific hardening recommendation does not make sense for a container (ex. Install CAC library to a container that does not allow person entities to connect using SSH for example), it is accepted to document these as non-applicable.
  - NIST 800-53v5 moderate controls plus FedRAMP+ IL 6 controls.
  - Risk Management Framework (RMF) process and required documentation.
  - Containers must be compliant with NIST SP 800-190 and Center for Internet Security (CIS) Docker Benchmark.
- Documentation
  - Generate and automate necessary documentation for Risk Management.
    - RMF Controls
    - Data Flows
- Enable TLS on all PaaS tools that have UI or send data. HTTP will be redirected and FIPS 140.2 utilized.
- Where personal-entity authentication and authorization are required, leverage DoD ADFS CAC authentication.
- Whenever possible, prohibit processes and containers from running as root.

5. APPENDIX C: SECURITY CONTROL INHERITANCE

PaaS/Platform Host providers rely on technical concepts such as process isolation, service routing, redundancy, firewall controls, and many other security layers. These typically align with DoD security concepts “out of the box”. Security overlays should always be leveraged (STIG, NIST 800-53, etc.). Ensure the selected host operating system has significant DISA STIG involvement, including security guidance/standards where available.

Leverage hardened Infrastructure as Code and Configuration as Code from Repo One whenever available to install various Kubernetes distributions supported by the Container Hardening team.

With a properly locked down hosting environment, containers inherit most of the security controls and benefits from infrastructure to host OS-level remediation requirements. The focus then shifts to container security requirements and application security requirements (e.g., static code analysis, unit testing, library CVE testing, etc.).

6. APPENDIX D: CONTAINER SECURITY/REMEDIATION CONSIDERATIONS

6.1 False Positives Commentary

The preferred approach to container security and remediation is to use tool standards such as Palo Alto Prisma/StackRox, Anchore, OpenSCAP, etc. If an OpenSCAP scan returns non-compliant result(s), always evaluate the validity of those findings. False positives are common within major host OS-based containers, as the security profiles normally account for all host-level controls potentially not applicable to a container build (e.g., GUI, CAC authentication, etc.). In addition to false positives, many of the base OS STIG requirements are not applicable in the containers either.

For false positives or non-applicable items, there are two options: either remediate to force the scanner to detect it as corrected or modify scanning profiles to mark specific components as non-applicable. For example, there are 363 configuration settings applicable to RHEL 7 base OS, and 93 of these controls are not inherited and/or applicable. If any of the 93 controls need to be remediated, 85 of them can be automated. Always automate all container-hardening processes leveraging CI/CD pipelines whenever possible. Manual processes should be avoided if possible. As stated previously, it is very important to select a proper base image. See Section 2.1 Container Hardening Pre-Process Steps.

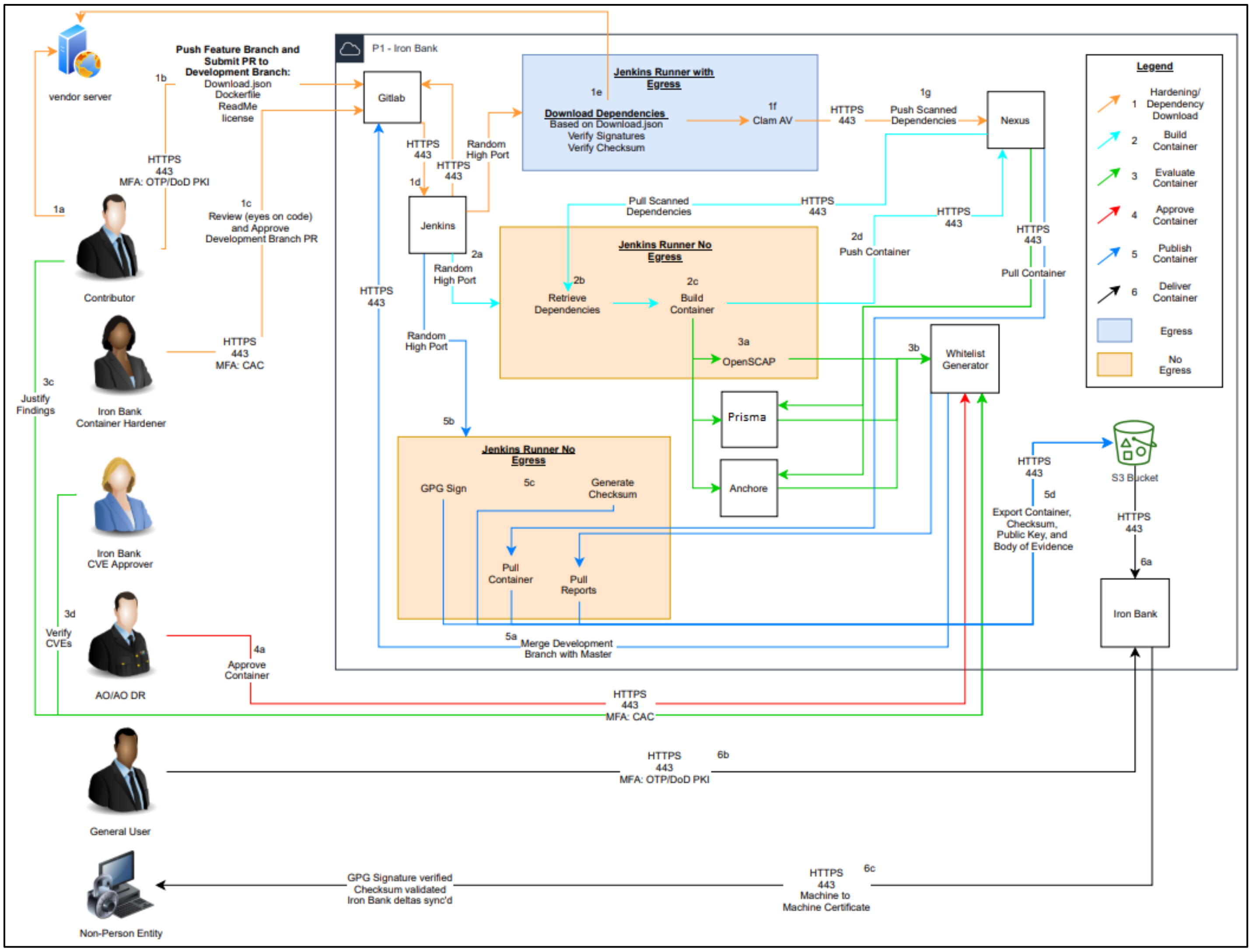
For example:

- If a container requires a base OS (such as RHEL/Alpine), the DoD hardened-base image of these OSs must be used.
- If a container needs OpenJDK such as a Java application or a COTS product using Java, the DoD OpenJDK base image must be reused whenever possible. This ensures that hardening of JDK is centralized and consolidated across all containers by leveraging inheritance with those base images.

For containers determined to be a true positive, always remediate when able.

7. APPENDIX E: VISUAL PROCESS FLOW

Figure 7-1: Visual Iron Bank Flow Diagram



7.1 Iron Bank Steps

1. Hardening/Dependency Download

a. Contributor updates application.

b. Contributor submits a feature branch to Gitlab. The branch will include the Download.json, Dockerfile, ReadMe, and a license. Once the branch is ready, the Contributor will submit a pull request to the Development Branch.

c. Iron Bank Container Hardeners will review the pull request with eyes on code. Once the hardeners validate that the pull request meets criteria specified, they will approve the pull request and merge the feature branch into the development branch.

d. The action of merging into the development branch will inform the CI server to start orchestrating the pipeline.

e. The first CI Runner will have egress to the Internet. It will look at the Download.json file to identify the components necessary to pull the contributors information into the environment securely. Once in the environment signatures and checksums will be validated to ensure providence.

f. Each item downloaded will be sent through a Clam AV scan. If there are threats identified, the download will be quarantined.

g. If there are no threats detected, dependencies will be pushed into a private Container Registry.

2. **Build Container**

   a. After dependencies have been validated, the CI server will start another CI Runner without any egress to perform the build operations.

   b. The runner will connect to the Container Registry and pull down the scanned dependencies.

   c. The runner will build the contributor's container without Internet access.

   d. After a successful build, the container will be pushed into the Container Registry

3. **Evaluate Container**

   a. After a successful build, the runner will execute an OpenSCAP/InSpec, Prisma/StackRox, and Anchore scans.

   b. Results of the scans will be uploaded to the Vulnerability Assessment Tracker (VAT)

   c. Contributor will connect to the VAT and justify any findings from the scans.

   d. Iron Bank CVE Approvers will review all the justifications submitted and validate the information as accurate and appropriate to satisfy the finding.

4. **Approve Container**

   a. With the findings of the scans validated, the AO or the AO Designated Representative (DR) will review the entire body of evidence and make the decision to approve the container.

5. **Publish Container**

   a. Once approved, the Vulnerability Assessment Tracker (VAT) will merge the Development Branch into the Master Branch.

   b. This will trigger the CI server to start a publish pipeline. Another CI Runner without egress will be started to perform the actions.

   c. The runner will pull the container in, sign the container, generate a checksum, and pull the body of evidence.

   d. The package will be pushed to multiple locations segregating the check sum and public key from the container and body of evidence.

6. **Deliver Container**
    a. The Iron Bank application will retrieve container information from the storage location utilized by the VAT.
    b. Users will be able to access the Iron Bank application and obtain the body of evidence and containers.
    c. Kubernetes clusters will have a container that connects to the Iron Bank or Registry One (new container registry) authenticating with a machine-to-machine certificate and synchronize containers.

8. APPENDIX E: CONTAINER BRANCHING STRATEGY

Figure 8-1: Container Branching Process

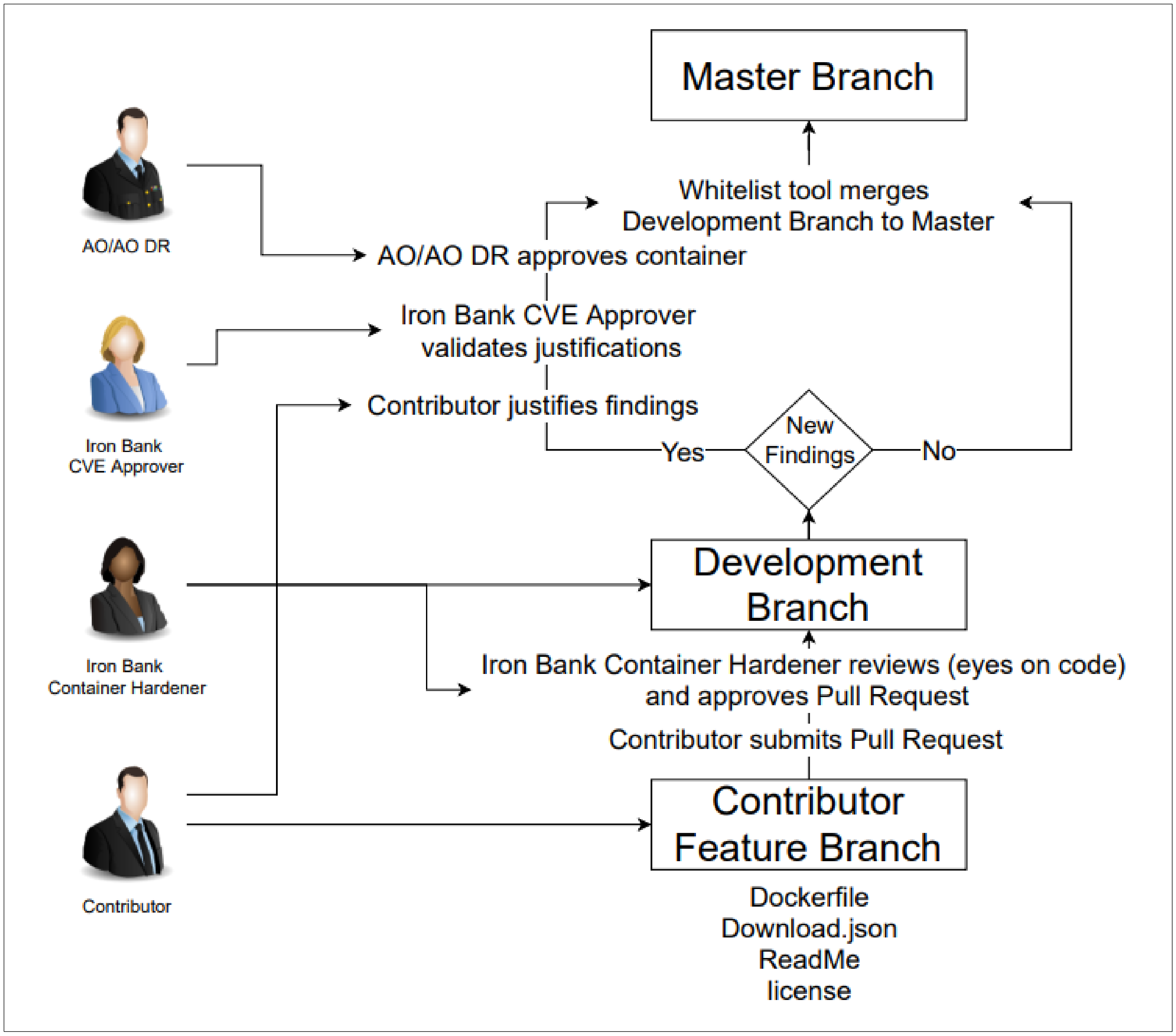
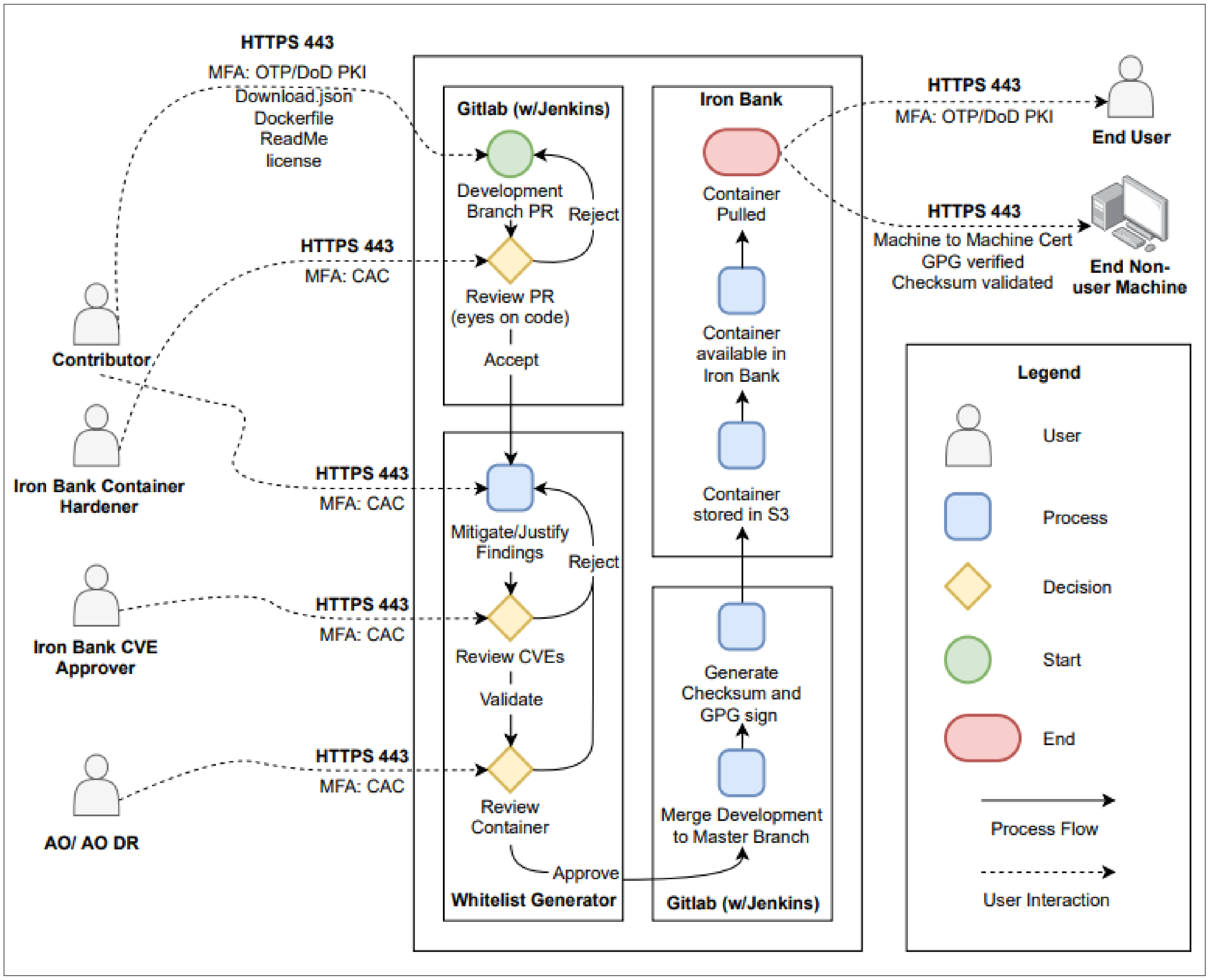

Master Branch
Whitelist tool merges Development Branch to Master
AO/AO DR
AO/AO DR approves container
Iron Bank CVE Approver validates justifications
Iron Bank CVE Approver
Contributor justifies findings
Yes
New Findings
No
Development Branch
Iron Bank Container Hardener
Iron Bank Container Hardener reviews (eyes on code) and approves Pull Request
Contributor submits Pull Request
Contributor Feature Branch
Contributor
Dockerfile
Download.json
ReadMe
license

Figure 8-2: Process Flow


HTTPS 443
MFA: OTP/DoD PKI
Download.json
Dockerfile
ReadMe
license
Contributor
HTTPS 443
MFA: CAC
Iron Bank Container Hardener
HTTPS 443
MFA: CAC
Iron Bank CVE Approver
HTTPS 443
MFA: CAC
AO/ AO DR
HTTPS 443
MFA: CAC
Gitlab (w/Jenkins)
Development Branch PR
Reject
Review PR (eyes on code)
Accept
Mitigate/Justify Findings
Reject
Review CVEs
Validate
Review Container
Approve
Whitelist Generator
Iron Bank
Container Pulled
Container available in Iron Bank
Container stored in S3
Generate Checksum and GPG sign
Merge Development to Master Branch
Gitlab (w/Jenkins)
HTTPS 443
MFA: OTP/DoD PKI
End User
HTTPS 443
Machine to Machine Cert
GPG verified
Checksum validated
End Non-user Machine
Legend
User
Process
Decision
Start
End
Process Flow
User Interaction